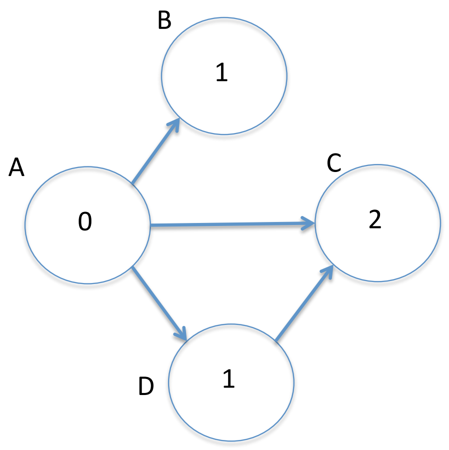
CS4320 Final Exam Solutions, Fall 2015

Part A) Map Reduce (10 points)

In this question, you are asked to provide Map Reduce pseudocode to number the nodes in a DAG (directed acyclic graph) in *topological order*.

You are given a DAG with no weights on the edges, and a designated source node. Your goal is to number all nodes in the graph such that the source has number 0, and for all nodes u, v, if there is an edge from u to v, then number(u) < number(v). In other words, for every node u, its number should be the length of the *longest directed path from the source to u*. Here is an example of a DAG and the numbers you should compute. Node A is the source; note that node C has number 2 even though it can be reached from the source in one edge; however, there is also a path A->D->C of length 2. Assume all nodes in your DAG can be reached from the source.



This will be an iterative Map Reduce algorithm. Assume the input to your mapper and the output to your reducer are the same, and have the format (nid, Node) where:

- nid is the key and is a unique node id for each node
- Node is the value; it is a data structure containing an adjacency list N.outneighbors (that contains the node ids of the node's outneighbors) and the number which you are computing, N.number.

<u>This question confused some students because they did not understand that "the output to your reducer" was intended to mean "the output OF your reducer". To make up for this we graded in a way that did not penalize you if you made this mistake.</u>

A.1) (2 points) State how N.number should be initialized for each node before any jobs are run.

For the source node N.number = 0, for all other nodes N.number = infinity or some other placeholder value.

A.2) (4 points) Give pseudocode for the Map portion of your solution. Solutions that do not use parallelism (such as processing the entire graph in a single mapper) will get little or no points. Be sure to make it very clear what the ouput format of your mapper is.

```
Map (nid n, Node N)
    distance = N.number
    emit (nid, N) //emit Node data structure
    if (distance != infinity)
        for (nid m in N.outneighbors)
            emit (m, d+1)
```

Output format is (nid, V) where nid is a node id and V is either an integer or the Node data structure (because we need to pass the data structure along in the graph)

A.3) (4 points) Give pseudocode for the Reduce portion of your solution.

```
Reduce (nid m, [d1, d2,… dk])
    maxDistance = infinity
    Node M = null
    for d in [d1, d2, … dk]
        if d is a Node then M = d
        else if (maxDistance = infinity OR d > maxDistance) then maxDistance = d
    M.number = maxDistance
    emit (m, M)
```

A.4) (1 point) What is the stopping condition, i.e. when will you stop running the iterative Map Reduce jobs?

You can stop when no numbers (distances) have changed in the entire graph in a MR iteration.

Part B) Concurrency Control (12 points)

B.1) (3 points) Consider a schedule S. Suppose we have two transactions T and T' in S. We say that T *precedes* T' if every action of T happens before every action of T', that is, there is no action by T that follows some action of T'. If T and T' are the only transactions in S, this means S is serial; however, if other transactions are in S as well, S does not need to be serial or even serializable.

Give an example schedule S and transactions T, T' in S such that T precedes T', S is conflict-serializable, but in any serial schedule that is conflict equivalent to S, T' is ordered before T. Explain your answer.

An example schedule is W3(A) W1(A) W2(B) W3(B). T1 precedes T2 in the schedule, but if you draw the conflict graph you will see the only serialization order is 2, 1, 3, so T2 must come before T1 in the conflict-equivalent serial ordering.

B.2) (3 points) Show two schedules involving the same transactions such that the schedules are view-equivalent to each other, but neither is view-serializable. Explain why they are view-equivalent to each other and why neither is view-serializable.

An example is R2(A) W1(A) W1(B) W2(A) and R2(A) W1(A) W2(A) W1(B). In both schedules T2 sees the initial value of object A. There are no other reads so the second condition for view-serializability does not apply. In both schedules, T1 is the final writer on B and T2 is the final writer on A. Thus they are view-equivalent. However, neither schedule is view-serializable as we would need to order 2 before 1 (due to the initial read) and also order 2 after 1 (because of the final write).

B.3) Consider a system that uses Optimistic Concurrency Control (OCC), specifically using the protocol discussed in class where the validation and write phase for each transaction occur in a critical section, i.e. only one transaction is in the validation/write phase at a time.

(2 points) Can a read-only transaction (a transaction that performs no writes) ever fail validation in Optimistic Concurrency Control? If yes, give an example, if no, explain why.

Yes, it can. Suppose T is read-only and reads the value of A. After the read but before T enters validation, some T' that wrote to A enters validation, passes validation and commits. Then T must fail validation because it should have seen the value of A written by T'.

(2 points) Can a write-only transaction (that performs no reads) ever fail validation? If yes, give an example, if no, explain why.

No, this is not possible; the only reason to fail validation is because of an intersection between the current transaction's read set and the write sets of previous transactions, so if the current read set is empty the validation will always succeed.

B.4) (2 points) State one similarity and one difference between MVCC and Snapshot Isolation.

Similarity: they both use versioning, they both allow reads to proceed without blocking

Difference: the readers see different versions of the items they read, there is a different criterion for checking and potentially aborting writers.

C) ARIES, Recovery and 2PC (9 points)

C.1) (3 points) Suppose we have an ARIES system and we are interested in reclaiming log space by garbage collecting log records that are no longer needed. At any point in ARIES execution, what is the oldest log record we need to retain? Give the LSN of this record as a function of relevant parameters in the system. Explain why we do not need to retain any log records earlier than the one you gave.

<u>This question confused some students because they assumed that "garbage collection" meant reclaiming portions of the log from memory rather than from disk. If you made this interpretation of the question, we did not penalize you in grading as long as your (memory-based) answer was correct.</u>

We always need to retain sufficient log records to perform an undo and a redo. For the redo, we need to keep logs as far back as the smallest recLSN in the DPT at the last checkpoint. For the undo, we need to keep logs as far back as the first update by any transaction that is currently in the system (has not committed, has not aborted or has aborted but has not been fully undone). Log records with LSNs smaller than the smaller of the two numbers mentioned above can be garbage collected.

C.2) (2 points) ARIES uses Strict 2PL on the pages, i.e. once a transaction has written to a page, no other transaction may write to the page until the original transaction aborts or commits. Explain why this is essential for correctness of ARIES. I.e, give a scenario where the above is not true and the recovery algorithm discussed in class is not able to correctly recover.

Suppose T1 writes to P1 and then before T1 commits T2 also writes to P1; moreover suppose the updates relate to the same portion of the page. Now T2 commits, and subsequently T1 aborts. The classical undo algorithm for T1 would undo T1's change by replacing the portion of the page that T1 modified with the "before-image" from T1's update log entry. However, this would cause T2's subsequent update to be lost as well.

C.3) (4 points) 2PC Presumed abort and 2PC presumed commit are both optimizations to 2PC; however there is an asymmetry, and commits in 2PC Presumed Commit must be handled differently than aborts in 2PC Presumed Abort. State what the difference is and explain why the different handling is necessary.

In 2PC Presumed Abort, in the abort case the coordinator does not need to force any log entries to disk. In 2PC Presumed Commit, in the commit case the coordinator needs to force both a begin entry upon the beginning of the protocol and a commit entry upon deciding to commit. This is so recovery can distinguish between the case where the crash happened before the coordinator decided to commit (and recovery should proceed backwards, i.e the transaction should abort) and the case where the crash happened after the coordinator decided to commit (and recovery should proceed forwards, committing the transaction).

Part D) (8 points) SQL and RA queries

Recall the following relational schema from the prelim: a database keeps track of nations (countries) as well as cities within each nation. You may assume all cities that exist in each nation are represented in the City table. The population fields represent the population of the nation and the city respectively and you may assume the database contains no null values anywhere. The *nid* in City is a foreign key referencing Nation.

Nation(<u>nid: integer</u>, name: string, population: integer)
City(<u>cid: integer</u>, name: string, population: integer, nid: integer)

D.1) (4 points) Write a query **in SQL** to find all nations containing exactly 3 cities. Display the nids and names of all such nations.

```
SELECT N.nid, N.name

FROM Nation N, City C

WHERE N.nid = C.nid

GROUP BY N.nid, N.name

HAVING COUNT(cid) = 3;
```

D.2) (4 points) Write a query **in Relational Algebra** to find all nations containing exactly 3 cities; display the nids and names. Note that this is exactly the same query as D1, except in RA.

The idea is to find all nations containing 3 or more cities, exclude from that set the nations that contain 4 or more cities, and return the nids/names of the remaining (non-excluded) cities. The query is:

$$\rho(C1, City), \rho(C2, city), \rho(C3, City), \rho(C4, city)$$

$$\rho(3OrMoreCities, \pi(C1.nid(\sigma_\alpha C1 \times C2 \times C3)))$$

$$\rho(4OrMoreCities, \pi(C1.nid(\sigma_\beta C1 \times C2 \times C3 \times C4)))$$

$$\pi_{nid,name}((3OrMoreCities - 4OrMoreCities) \bowtie Nation)$$

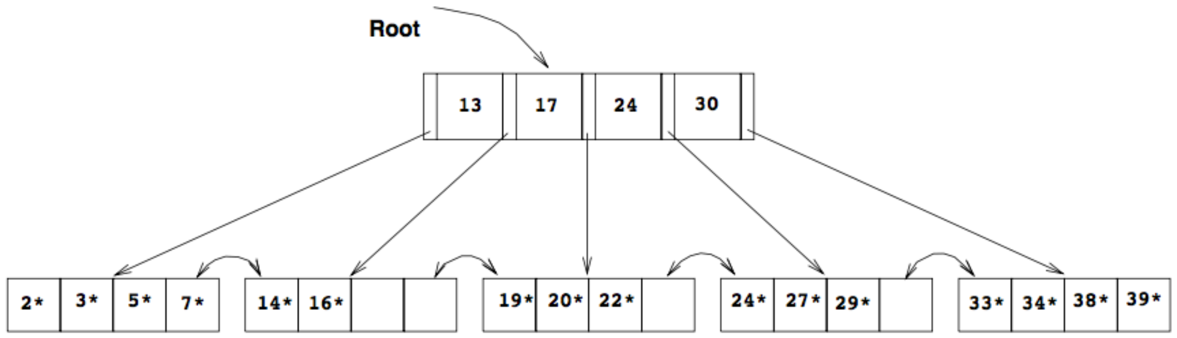
Where the selection conditions are as follows:

α is $C1.nid = C2.nid \wedge C2.nid = C3.nid \wedge C1.cid \neq C2.cid \wedge C1.cid \neq C2.cid \wedge C1.cid \neq C3.cid$

β is $C1.nid = C2.nid \wedge C2.nid = C3.nid \wedge C3.nid = C4.nid \wedge C1.cid \neq C2.cid \wedge C1.cid \neq C2.cid \wedge C1.cid \neq C4.cid \wedge C2.cid \neq C3.cid \wedge C2.cid \neq C3.cid \wedge C3.cid \neq C4.cid$

Part E) (8 points) Indexing

Consider the tree shown below.



E1). (4 points) Give a sequence of five entries such that inserting them and then deleting them in reverse order will result in the original tree (ie we insert e1, e2, e3, e4, e5, then delete e5, e4, e3, e2, e1). If you need to make any assumptions about how we handle insertions/deletions state them clearly. Explain why we get the original tree back after the five deletions.

An example sequence is 17, 18, 13, 15 and 25. Inserting 17 then 18 causes the third leaf to split, leading to a root split and the tree gains a level. The remaining insertions don't change the shape of the tree, and neither do the deletions (25, 15, 13). When 18 and 17 are both deleted the third and (new) fourth leaves of the tree are merged and one of the children of the root becomes underfull, with no redistribution possible. Thus the tree shrinks by one level and we have the original tree back.

E1). (4 points) Give a sequence of five entries such that inserting them and then deleting them in reverse order will result in a different tree (ie we insert e1, e2, e3, e4, e5, then delete e5, e4, e3, e2, e1). Draw the final tree after the 5 entries have been inserted and deleted.

One example sequence is 13,15,23,30,40. The last insert causes the last leaf to be split; say it is split into leaves containing (33, 34) and (38,39, 40). The root is split and the tree grows one level; the level above the leaves contains (13,17) and (30, 38) while the root contains (24). When 40 is deleted, the new leaf node is not underfull and the shape of the tree does not change. Deleting the remaining nodes in reverse order also does not change the shape of the tree.

Part F) (16 points total, 2 points each) Answer the following in no more than a few sentences. Excessively long answers will be penalized even if correct.

F.1) Give two differences between OLTP and OLAP workloads.

OLTP workloads contain both reads and writes, consist of predefined (not ad-hoc) queries/updates, typically access few tuples in the database. OLAP queries are read-only, access a lot of tuples and are often ad-hoc

F.2) How can a B+-tree be made *cache-conscious* and why is this a good idea?

A cache-conscious data structure makes use of the memory hierarchy above RAM to increase performance. In the case of B-trees, this involves compressing each node as much as possible so each node can fit in a single cache line. This makes search inside the node very fast

F.3) Give one difference between the BSP model and Map Reduce.

In BSP, there is no intermediate sort-shuffle and no requirement to follow a key-value data model. Also, not every node needs to compute at every step.

F.4) Draw a kd-tree for the following set of points: (2,3), (5,4), (9,6), (4,7), (8,1), (7,2). Start with the x dimension for splitting, and choose the median point along each dimension when splitting. If you need to make any assumptions, state them. Be sure to draw the actual tree produced.

Multiple correct answers are possible, including the one below:

```
                (7,2)

                /      \

        (5,4)          (9,6)

        /      \       /

(2,3)          (4,7) (8,1)
```

F.5) Consider the semijoin and Bloomjoin distributed join algorithms. Give one advantage of each one over the other.

The semijoin is more precise (only tuples that match are shipped) but the Bloomjoin requires shipping less data to the site with the other relation (we ship only a Bloom filter which is a bit vector, rather than the entire projection of a relation).

F.6) What is a key-value store?

It is a datastore that supports only a get/put/delete API for the (key,value) pairs it contains. (Minor variations/enhancements of the API are possible).

F.7) Give an example of a relation that is in 3NF but not in BCNF. Explain your answer.

SBDC with S->C and C->S. Then CBD and SBD are both keys and every nontrivial FD has a portion of a key on the RHS, so we are in BCNF. However we are not in BCNF as the LHS of C ->S (or S ->C) is not a superkey.

F. 8) What is a weak entity? What serves as the primary key of a weak entity?

In an ER diagram, a weak entity represents a concept that does not have a primary key of its own but is uniquely identified by the primary key of its parent (or owner) entity together with a partial key (a set of attributes of the weak entity itself).